VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

WEBOVÁ APLIKACE S INTERAKTIVNÍMI VZDĚLÁVACÍMI VIDEI

WEB APPLICATION WITH INTERACTIVE EDUCATIONAL VIDEOS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JIŘÍ TOBIÁŠ**
AUTHOR

VEDOUCÍ PRÁCE **Doc. Dr. Ing. DUŠAN KOLÁŘ,**
SUPERVISOR

BRNO 2017

Zadání bakalářské práce/20015/2016/xtobia02

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Tobiáš Jiří**
Obor: Informační technologie
Téma: **Webová aplikace s interaktivními vzdělávacími videi**
Web Application with Interactive Educational Video
Kategorie: Web

Pokyny:

1. Nastudujte možnosti interaktivních videí v HTML5 (h5p.org).
2. Navrhněte webovou aplikaci obsahující: evidenci uživatelů; nahrávání a správu interaktivních videí; interaktivní školení přihlášeného studenta (skoky ve videích spolu s textem, vyhledávání, testy, atd.); analytické a přehledové funkce.
3. Po konzultaci s vedoucím navrženou aplikaci včetně interaktivního prohlížení videí implementujte.
4. K aplikaci vytvořte návod a dále vytvořte vzorové interaktivní video i s otázkami a anotacemi, aplikaci důkladně otestujte na všech kategoriích uživatelů.
5. Zhodnoťte použitelnost aplikace a navrhněte další rozšíření.

Literatura:

- [online]: H5P - Create and Share Rich HTML5 Content and Applications, https://h5p.org/, cit. 2016-09-26
- Dle doporučení vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- První 2 body.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kolář Dušan, doc. Dr. Ing.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Cílem této práce je navrhnout a vytvořit webovou aplikaci, pro správu interaktivních výukových materiálů. Aplikace bude schopna nabízet uživatelům dostupná videa, která mají přístupná (viditelná) v rámci zapsaných kurzů. Dále bude zobrazovat uživatelům výsledek (hodnocení) jejich aktivit. Pro správce materiálů bude aplikace nabízet možnost správy kurzů, materiálů, uživatelů a také bude vytvářet podrobnější výstupy, které by měly sloužit jako zpětná vazba, pro zlepšení samotného obsahu. Aby bylo možné vytvářet takovéto výstupy, bude muset být aplikace schopna zaznamenávat jednotlivé uživatelské interakce.

Abstract

Goal of this work is create web application for the management of interactive education contents. The application will be able to show interactive content for users in course. Also, it will show users the result of their activities. For content administrators, the application will provide more detailed outputs that should serve as feedback, to improve the content. For creating outputs, the application will need to be able record users activities.

Klíčová slova

Interktivní video, webová aplikace, Google web toolkit, Java, Javascript, framework, H5P, MySQL, GWT, Sencha GXT, Ujorm

Keywords

Interactive video, web application,Google web toolkit, Java, Javascript, framework, H5P, MySQL, GWT, Sencha GXT, Ujorm

Citace

TOBIÁŠ, Jiří. *Webová aplikace s interaktivními vzdělávacími videi.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Doc. Dr. Ing. Dušan Kolář,

Webová aplikace s interaktivními vzdělávacími videi

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Doc. Dr. Ing. Dušana Koláře. Další informace mi poskytl Ing. Radek Majer z firmy Tritius Solutions a.s. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Tobiáš
18. května 2017

Poděkování

Tímto bych chtěl podekovat Doc. Dr. Ing. Dušanovi Kolářovi za odbornou pomoc a cenné rady pro psaní této bakalářské práce a také bych chtěl poděkovat Ing. Radku Majerovi za odborné rady ohledně použitých technologií.

Obsah

Kapitola 1

Úvod

Tato práce ze zabývá tvorbou webové aplikace pro podporu online výuky a školení pomocí interaktivních videí. Cílem je nastudovat možnosti interaktivních videí v HTML5. V dalším kroku vytvořit návrh aplikace zajišťující správu uživatelů a materiálů. Dalším požadavkem jsou analytické a přehledové funkce pro jednotlivé výukové materiály. V posledním kroku pak implementovat a otestovat výslednou aplikaci.

1.1 Motivace

V současné době existují aplikace, často v podobě SaaS[1], které umožňují přidávání, prohlížení nebo úpravu interaktivních materiálů a následné sdílení mezi uživateli. Problémem však je fakt, že takovéto aplikace jsou často zpoplatněné a jejich verze zdarma je velmi ochuzená o důležité funkce.

Druhou možností jak publikovat své výukové materiály, je využití některých redakčních systémů, jako například WordPress, Drupal, Joomla. Konkrétně pro tyto tři systémy existuje zásuvný modul (plugin) H5P, který je pro vytváření interaktivních materiálů výbornou volbou. Ovšem, při použití takovéhoto řešení je potřeba uvědomit si, že zásuvný modul má minimální implementaci na serveru a tedy neuchovává informace o uživatelských aktivitách.

Posledním výrazným problémem je lokalizace současných řešení. V českém nebo slovenském jazyce se takovéto aplikace nevyskytují.

Z tohoto důvodu vznikla také tato práce, která by měla výše zmíněné problémy odstranit a v případě lokalizace by měla nabídnout vícejazyčné rozhraní.

[1] Software as a Service, je způsob využívání aplikace jako službu, za kterou uživatel platí poplatek.[31]

Kapitola 2

Definice pojmů

Součástí textu je několik pojmů, které je potřeba pro porozumění textu definovat. V této kapitole budou tyto pojmy vysvětleny.

2.1 Interaktivní obsah

Základem tohoto pojmu je slovo "interaktivní", které nemusí vždy nabývat stejného významu. V našem případě budeme rozumět pojmem "interaktivita" jakousi aktivitu uživatele s technickým zařízením. Takovéto zařízení je schopné registrovat uživatelské akce, dále je vyhodnocovat a následně reagovat nějakým způsobem na tyto podněty. Typické uživatelské akce pro takový interaktivní systém je například stisknutí klávesy, pohyb myší atd.

2.2 Aplikační rámec

V informačních technologiích se aplikační rámec (někdy jenom rámec) definuje jako soubor knihoven, nástrojů a konvencí, které si kladou za cíl zjednodušit, zrychlit a zpřehlednit vývoj softwaru. [2]

2.3 Otevřený software

Často se setkáváme s anglickým výrazem *open source*. Zjednodušeně se jedná o typ licence, který umožňuje volně stahovat, upravovovat a sdílet software, jenž je pod touto licencí vytvořen.[22]

2.4 Uživatelské rozhraní

Uživatelské rozhraní (*User Interface - UI*) zajišťuje komunikaci mezi uživatelem a určitým systémem. Cílem takového uživatelského rozhraní je umožnit uživateli vykonávat požadované úlohy s obsluhujícím systémem efektivně a rychle. Velmi známé a dříve velmi používané *CLI - Command Line Interface* rozhraní vyžívající k interakci (se systémem) příkazové řádky. Příkladem *CLI* je například systém MS-DOS. [24]

2.5 Grafické uživatelské rozhraní

Grafické uživatelské rozhraní (*Graphical User Interface - GUI*) je typ uživatelského rozhraní, které využívá grafických prvků k dosažení intuitivního ovládání daného systému. Mezi typické grafické prvky, které GUI využívá, patří tlačítka, okna, ikony, obrázky atd.[24]

Kapitola 3

Možnosti interaktivních videí v HTML5

Kapitola popisuje služby a aplikační rámce, které lze použít pro tvorbu, správu nebo sdílení interaktivních videí.

Aplikační rámec **H5P** je rozsáhlý soubor knihoven pro vytváření nejen interaktivních videí, ale jiných interaktivních materiálů (prezentace, obrázky atp.). Knihovny jsou především navrhnuté pro integraci s tzv. CMS (Content Management System). To jsou systémy pro správu obsahu. Mezi nejznámější CMS systémy patří Wordpress, Joomla a Drupal. H5P pro tyto systémy vytvořilo plugin (zásuvný modul), který umožňuje správu a publikaci interaktivních materiálů.

Další možností jak používat H5P pro vytváření interaktivního obsahu, je využít otevřenosti kódu a použít připravenou implementaci knihoven v jazyce PHP.

Nevýhodou těchto dvou řešení je jejich jednoduchost. Prakticky veškerou obsluhu vykonává JavaScript ve webovém prohlížeči a jakákoliv podpora ze strany serveru není implementována. Podporou ze strany serveru mám na mysli především ukládání správných nebo chybných odpovědí, ukládání a správa otázek atd. Pro požadovaný systém je nutné přidat tuto funkcionalitu.[8]

Výhodou tohoto rámce je množství připravených knihoven pro tvorbu interaktivních materiálů, svobodná licence a možnost vytvářet jednoduše vlastní zásuvné moduly. Možnosti tohoto rámce je možné zdarma vyzkoušet na [6].

Další možností pro tvorbu interaktivních videí je využít existujicích řešení. Jedním z existujících řešení je webová aplikace **Playposit** [16], která svojí funkcionalitou téměř splňuje požadovaná specifika výsledné aplikace. Je možné si zde založit vlastní třídu, dále do ní přidávat žáky. Celé třídě je možné zpřístupňovat videa s interaktivním obsahem. Výstupem poté mohou být statistiky jednotlivých otázek, grafy atp. Pro samotná videa je možné nastavit datum, do kterého je možné video shlédnout. Silnou stránkou aplikace je velké množství přiložených nástrojů pro práci s video obsahem. Ovšem nevýhodou tohoto řešení je spousta omezení v bezplatné verzi.

Kapitola 4

Programovací jazyky

V následujících kapitolách se setkáme s technologiemi, které využívají, resp. jsou implementované v konkretním programovacím jazyku. Tato kapitola popisuje tyto jazyky.

4.1 Java

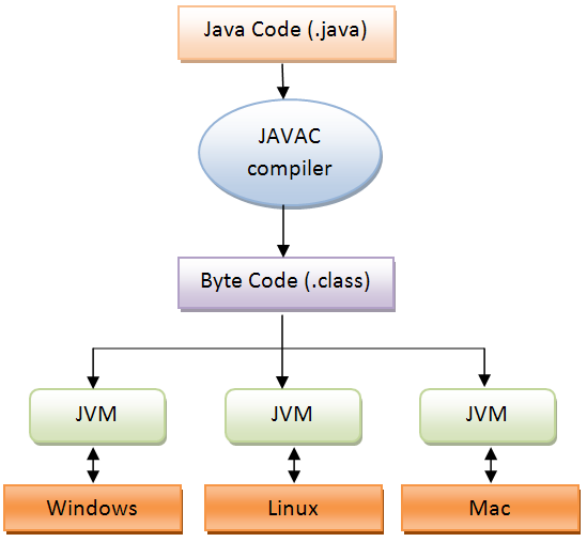
Programovací jazyk Java byl představen v květnu roku 1995 firmou Sun Microsystems. Jazyk Java používá podobnou syntaxi jako velmi známé jazyky C a C++. To přinášelo vývojářům, kteří měli zkušenosti s touto syntaxí, jednodušší učení konstrukcí tohoto jazyka a následný přechod do jazyka Java.

Hlavními rysy jazyka Java jsou: **Jednoduchost** – syntaxe podobná dříve známým jazykům C a C++. Většina nízkoúrovňových konstrukcí zde byla odstraněna. **Nezávislost na architektuře** – Java není závislá na určité architektuře nebo operačním systému. **Objektově orientovaný** – téměř vše je zde reprezentováno jako objekt. **Bezpečnost** – poskytuje bezpečné prostředky pro tvorbu internetových aplikací. **Přenositelnost** – programy psané v Javě lze spouštět v jakémkoliv prostředí, kde je existuje běhové prostředí Javy, tzv. *JRE (Java Runtime Environment)*. **Interpretovaný** – Java kód je nejprve přeložen do bajtkódu, což je forma instrukční sady navržená pro snadnou přenositelnost aplikace. Ten je následně za běhu aplikace prováděn. Pro interpretaci Java kódu se využívá tzv. virtuální stroj Javy (Java Virtual Machine, JVM), viz obrázek 4.1. Přesto, že je zde použita interpretace kódu, zachovává si Java vysokou výkonnost. Toho je docílené především různými optimalizacemi a také možnosti tzv. *JIT (Just In Time)* kompilace, kdy se bajtkód přeloží přímo do strojového kódu.[33]

V dnešní době je tento jazyk velmi rozšířený. Firma **Oracle**, která je současným vlastníkem oficiální implementace standardní Java platformy (Java SE), uvádí, že tato platforma běží na 15 miliardách zařízení.[3]

Uplatnění najdeme především v mobilních telefonech, například v zabudovaných zařízeních, nebo v desktopových a webových aplikacích.

Java Enterprise Edition, nebo také Java EE označuje vývojovou platformu pro vytváření komplexních podnikových aplikací. Java EE je nadstavbou nad Java SE, ke které přidává několik aplikačních rozhraní. Tato rozhraní se zaměřují na usnadnění vývoje jednotlivých částí nebo vrstev aplikace.



Obrázek 4.1: Znázornění architektury JVM. Převzato z [10]

4.2 JavaScript

Javascript je velmi populární programovací jazyk, který se nejčastěji provozuje na straně webového prohlížeče. Javascript je interpretovaný, multiplatformní programovací jazyk. Syntaxí připomíná dříve zmíněné C, C++ nebo jazyk Java a umožňuje využívat některé principy objektově orientovaného návrhu.

Dále je dobré zmínit fakt, že se jedná o dynamicky typovaný jazyk. Typová kontrola se provádí až při vykonávání programu a pokud je to možné, lze proměnnou dynamicky přetypovat. Tato vlastnost dělá samotný program „pružnější“, avšak je zde riziko typové nekompatibility, která se projeví pouze za běhu aplikace.

JavaScript se používá především pro vytváření webových stránek a aplikací, jelikož umožňuje manipulaci s HTML prvky na základě nějaké akce, například po interakci s uživatelem. Existuje mnoho nadstaveb a knihoven, které urychlují a usnadňují vývoj v jazyku JavaScript, mezi které se řadí velmi známí *jQuery*, *Angular Js*, *React* atp. Můžeme se s ním setkat při programování senzorů, robotů, nebo v kontextu s *IoT*[1].[30]

Přestože jazyk JavaScript je často využíván pro vytváření GUI, lze ho použít i pro logiku na straně serveru. Jedním z aplikačních rámců využívající tohoto přístupu je *Node.js*.

[1]Internet věcí - anglicky „Internet of things“ je označení pro propojování různých zařízení se sítí Internet

Kapitola 5

Návrh aplikace

5.1 Databáze

Pro znázornění databázové struktury budeme využívat ER diagramů.

5.1.1 ER diagram

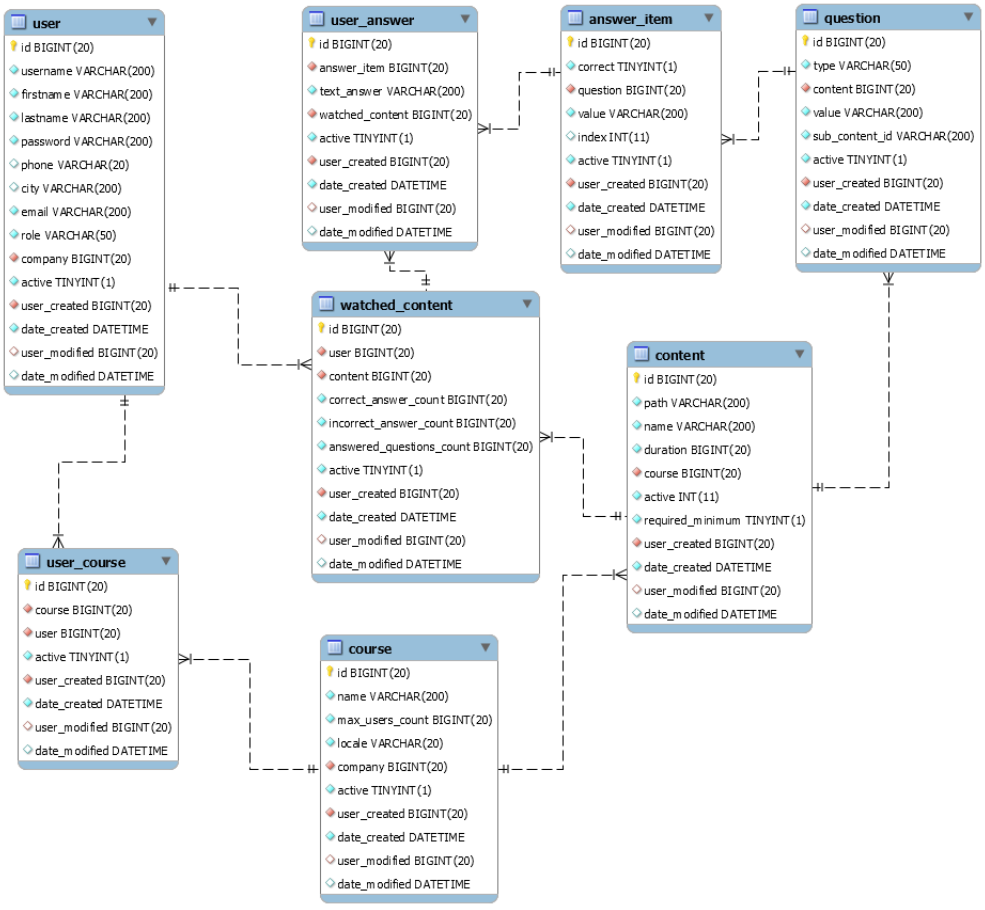
ER (Entity-Relationship) diagram popisuje data a často se používá pro popis relačních, objektově-relačních nebo objektových databází. Znázorňuje schéma dat a jejich vazby, ale dále už nepopisuje možné operace s entitami.[27]

Entita - popisuje reálný objekt, z čeho se skládá a jeho vlastnosti pro nás důležité.
Relace - vztahy mezi entitami.
Atribut - jedna vlastnost entity. Entita se zpravidla skládá z více atributů.

Popis klíčových entit

V této sekci bude popsán výběr klíčových entit databáze a některých jejich atributů, které jsou pro fungování systému naprosto nepostradatelné. Pro znázornění vazeb mezi entitami bude přiložen grafický návrh databáze ve formě ER-diagramu, viz 5.1.
user – entita představující uživatele systému. Všichni uživatelé systému musí mít záznam v databázi. Požadované atributy entity jsou: jméno, příjmení, login, email, heslo (do databáze se ukládá v zašifrované podobě) a uživatelská role (např. učitel, administrátor, student atd.). U uživatele je také možné ukládat nepovinné atributy, příkladem může být bydliště, telefonní číslo.
course - tato entita zachycuje kurz, který si může uživatel zapsat. Důležité atributy kurzu pro systém je název kurzu a jeho jazyková lokalizace.
content - velmi významná entita, jež uchovává informace o samotném obsahu, kterým může být například interaktivní video. Podstatným atributem entity je cesta k obsahu, neboli kde se obsah fyzicky nachází na disku. V případě této entity je dobré ještě zmínit atribut *required_minimum* reprezentující minimální požadovanou úspěšnost pro splnění daného výukového materiálu.
question - otázka přiřazená k obsahu. Pro statistické potřeby je nezbytné uchovávat některé informace o otázkách, jež jsou v materiálech zahrnuty. V prvé řadě si budeme ukládat textovou hodnotu otázky, typ a materiál, neboli obsah, ve kterém se otázka vyskytuje.
answer_item - definice možností na danou otázku. Pro statistické účely nezbytná. Díky in-

formacím uložených v této entitě je možné rozhodovat o správnosti uživatelských odpovědí. Atribut *correct* je právě tím důležitým nositelem informace o správnosti dané možnosti.
watched_content - každé zhlédnutí, nebo spuštění materiálu budeme zaznamenávat pomocí entity *watched_content*, volně přeloženo „zhlédnutí obsahu“. Tím budeme schopni uchovávat historii prohlížení obsahů a uživatelský pokrok při odpovídání na otázky. Entita obsahuje atributy jako *correct_answer_count*, *incorrect_answer_count*, nebo *answered_question_count*. V překladu „počet správných odpovědí“, „počet chybných odpovědí“ a „počet zodpovězených otázek“. Tyto atributy budou sloužit pro dříve zmíněné statistické údaje.
user_answer – jedná se o entitu představující konkrétní odpověď uživatele na konkrétní definici odpovědi (*answer_item*) a také pro určité zhlédnutí videa. Nejvíce významným atributem je zde *text_answer* - textová reprezentace odpovědi.
user_course - pouze pomocná entita odstraňující tzv. „M-ku N“ relace mezi uživatelem (*user*) a kurzem (*course*).

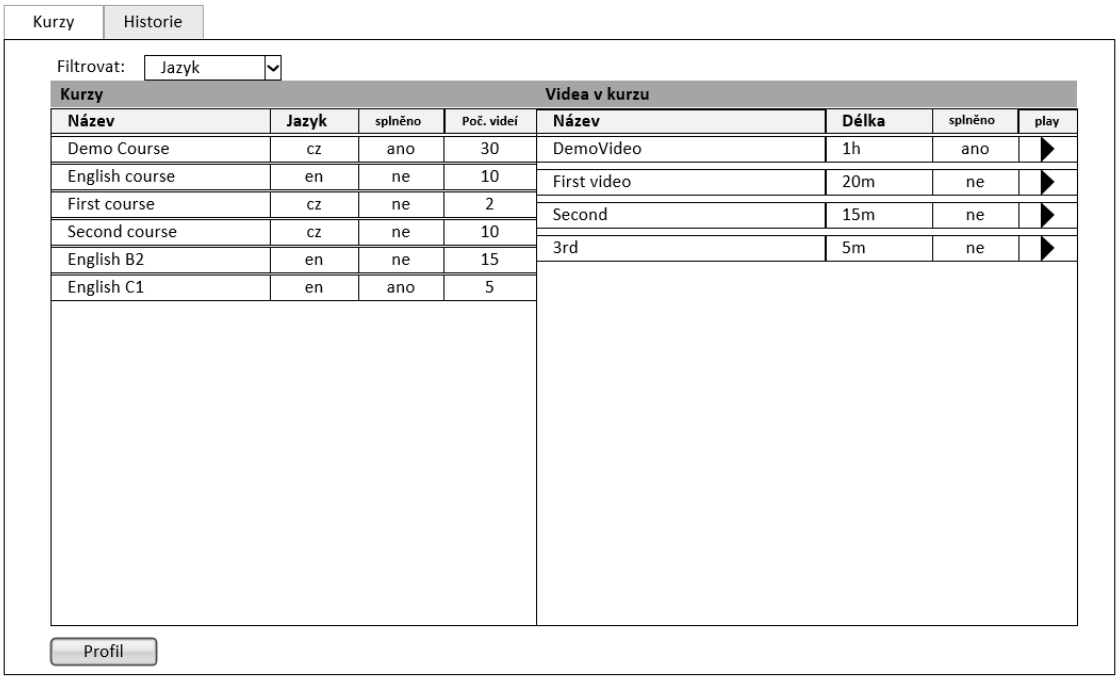


Obrázek 5.1: ER diagram aplikace

5.2 Grafické uživatelské rozhraní

Pro aplikaci je zapotřebí zobrazovat kurzy, které má uživatel zapsané. Umožnit lektorům a administrátorům spravovat kurzy, tj. přidávat uživatele (studenty/školící zaměstnance) do kurzů, zobrazovat jejich dosažené výsledky. Grafické uživatelské rozhraní musí respektovat uživatelská oprávnění a zobrazovat data na která má přihlášený uživatel práva. Důležitou vlastností uživatelského rozhraní je podpora více jazykových lokalizací a možnost přenastavení jazyka.

Návrh pro editaci a zobrazení informací je demonstrován na obrázku 5.2. Návrh pro zobrazení statistických dat v podobě grafů je demonstrován na obrázku 5.3.



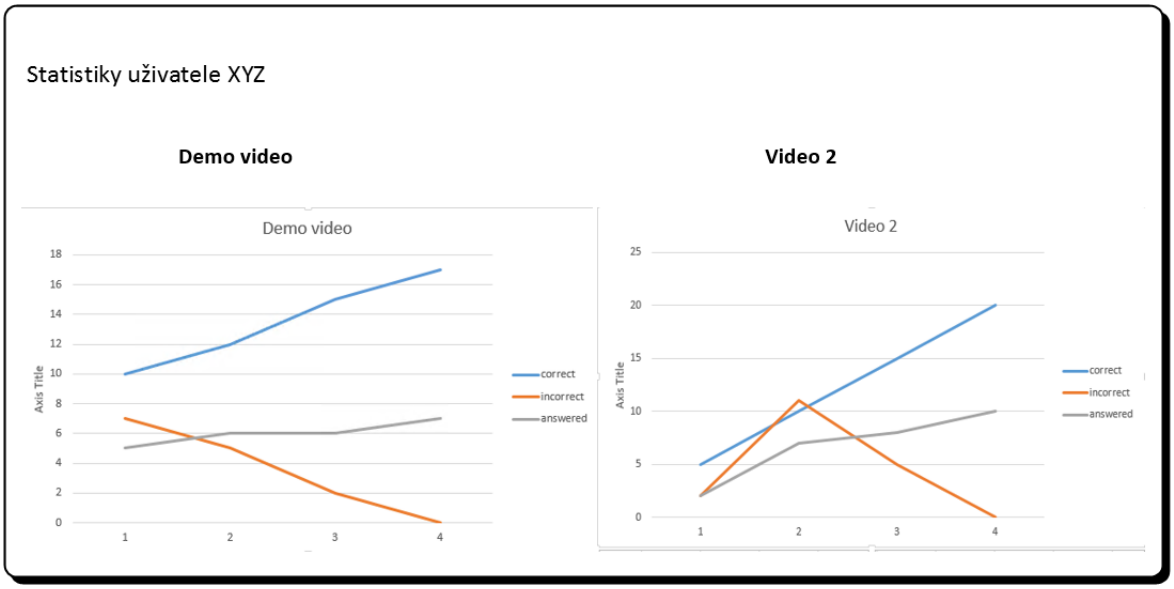
Obrázek 5.2: Návrh grafického uživatelského rozhraní pro zobrazení kurzů, které má uživatel zapsané

5.3 Uživatelské role

V rámci systému jsou uživatelská oprávnění definována pomocí uživatelských rolí. Tato oprávnění definují jaký obsah se má uživateli zobrazit a jaké akce má povolené provádět. V systému existují tyto tři role:

User - základní uživatelská role s minimálními právy. Umožňuje zobrazit zapsané kurzy, videa v kurzu a vlastní historii s procentuální úspěšností. Příkladem reálného uživatele systému, kterému je přiřazena tato role, může být student.

Teacher - zde je přidána možnost organizovat kurzy, uživatele (pouze typy *User*) v kurzu, přidávat a spravovat videa, zobrazovat statistiky a historii uživatelů. Typickým příkladem z reálného prostředí je učitel nebo lektor.



Obrázek 5.3: Návrh grafického uživatelského rozhraní pro zobrazení statistických údajů uživatele

Admin - plný přístup k systému. Oproti roli *Techer* má možnost spravovat a zobrazovat všechna data.

Kapitola 6

Architektura a použité technologie

Kapitola popisuje architekturu a technologie, které lze použít pro vývoj požadované aplikace a dále podrobněji popisuje vybrané technologie použité při vývoji výsledné aplikace. Součástí kapitoly je vlastní návrh výsledné architektury a popis jak obecných vrstev vycházejících z některých vzorů, tak popis částí rozšiřující některou z vrstev.

6.1 Síťová architektura

Z hlediska síťové architektury je systém postaven na architektuře klient-server, kde server je mozek systému a bývá umístěn na výkonném hardwaru. Server zpracovává požadavky od klientů a následně provádí požadované akce, poté vrací výsledek na klienta, kde je následně zpracován (například vykreslením dialogu). Server je implementován v jazyce Java EE verze 7.

Naopak klient, často interpretován pomocí prohlížeče a nemá přímý přístup k datům, pokud nějaká data potřebuje, musí požádat server, který mu požadovaná data zpřístupní (pokud je to možné).

6.2 Vícevrstvá architektura

Vícevrstvá architektura rozděluje aplikaci do nezávislých vrstev, které spolu komunikují a vytváří tak celistvý program. Nejjednodušší vícevrstvou architekturou je třívrstvá architektura, která se skládá z datové, aplikační a prezentační vrstvy.

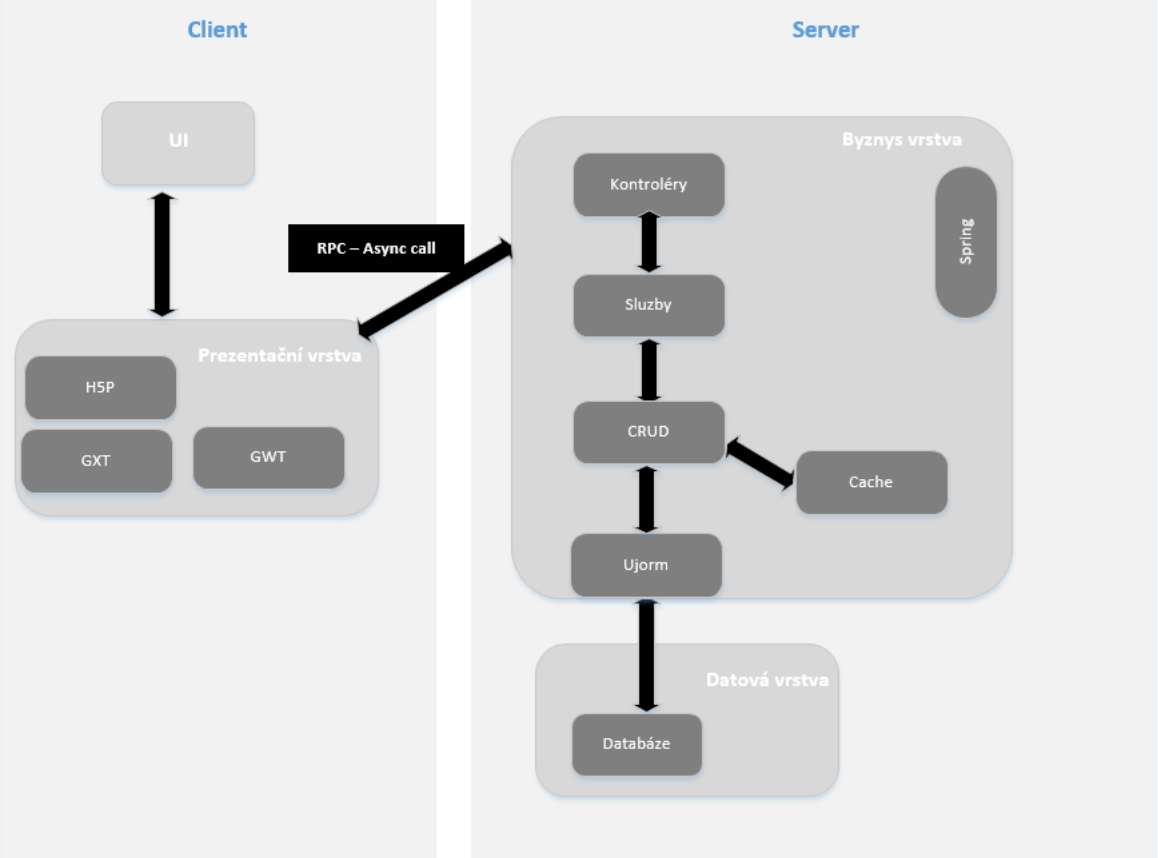
Při návrhu architektury aplikace jsem vycházel z vícevrstvé architektury a její struktura je znázorněna na obrázku 6.1.

6.2.1 Datová vrstva

Pro ukládání dat se nejčastěji využívají relačních databází. Pro požadovanou aplikaci jsem si vybral jedny z nejpoužívanějších databázových systému, MySql a MsSql.

`MySQL` - relační databázový systém, jenž umožňuje jazyku `Java` (ale i jiným technologiím) spolupracovat na zpřístupnění a zobrazení dat v požadovaném, nejčastěji čitelném formátu. Jedná se o server zpracovávající dotazy ve strukturovaném dotazovacím jazyce `SQL`[1] a je navržený pro zpracování velkého množství složitých dotazů. Jelikož je MySQL

[1] Structured Query Language



Obrázek 6.1: Struktura výsledné aplikace

relační databázový systém, umožňuje spojování mnoha různých tabulek a díky tomu nabízí maximální efektivitu a rychlost. Patří mezi nejvíce rozšířené databázové systémy a to především díky své `open source` licenci.[28]

`MsSQL` - také relační databázový systém určený především byznys účelům. Využívá dotazovacího jazyka `Transact-SQL`, což je upravená verze `SQL` firmou *Microsoft*, která stojí za vývojem systému.[29]

Jelikož databázové systémy musí zpracovávat mnoho operací (transakcí), je nezbytné zajistit neporušitelnost (integritu) databáze. Aby se zajistilo integritě databáze, musí transakce splňovat tzv. **ACID** vlastnosti.[34]

Atomičnost (**A**tomicity) transakce znamená, že transakce je jako operace dále nedělitelná. Buď jsou provedeny všechny jednotlivé operace, ze kterých se transakce skládá, nebo není provedena žádná z nich. Pokud dojde k poruše při provádění transakce, je potřeba zajisti návrat do stavu před zahájením transakce.

Konzistence (**C**onsistency) transakce znamená, že transakce, která je izolovaná (viz. *Izolace*), neporušuje konzistenci databáze. Pokud je databáze před zahájením provádění transakce konzistentní, musí být konzistentní i po ukončení provádění.

Izolace (**I**solation) transakce znamená, že veškeré souběžně probíhající transakce budou z hlediska provádění operací s daty v databázi izolované. Pro každou dvojici transakcí se jeví, že transakce „A“ skončila dříve než transakce „B“ zahájila svoji činnost nebo transakce

„A“ zahájila svoji činnost až poté, co transakce „B“ ukončila svoji činnost. Výsledný efekt provádění operací s daty se pak jeví jako sekvenční, namísto souběžných.

Trvalost (**D**urability) transakce znamená, že všechny změny, které transakce provede jsou trvalé. Tj. i po výpadku systému je možné registrovat změny, které transakce úspěšně provedla. Tato vlastnost předchází nechtěným ztrátám dat.[34]

6.2.2 Byznys vrstva

Byznys vrstva, nebo také vrstva logiky, je jádro celé aplikace. Obsahuje veškerou logiku aplikace a provádí potřebné výpočty. Je zodpovědná za přenos dat mezi vrstvou dat a prezenční vrstvou.

V rámci návrhu výsledné aplikace jsem tuto vrstvu rozdělil na dílčí logické části. Rozdělení vrstvy na jednotlivé části je znázorněné na obrázku 6.1.

Připojení k databázi

Připojení k databázi aplikace využívá *JDBC*[2] ovladače, což je aplikační rozhraní pro jazyk Java, které definuje jednotné rozhraní pro přístup k relačním databázím.

Přístup k datům

Pro přístup k datům aplikace používá aplikační ORM[3] rámec Ujorm. **Ujorm** je rámec napsaný v jazyce `Java` a je určený pro rychlý vývoj Java aplikací využívajících relační databáze. Ujorm je silně typovaný rámec založený na generikách, díky čemuž je `Java` překladač schopen odhalit typové chyby už v době překladu. S tím také souvisí fakt, že kód je možné bezpečně refaktorovat[4] a také napomáhá ke snadnějšímu vytváření generických komponent.[32]

CRUD

Vrstva CRUD, je zkratkou pro čtyři základní operace se záznamy v databázi, které zajišťuje. Vytváření (create), čtení (read), editace (update) a mazání (delete).

Pro zrychlení běhu aplikace je k této vrstvě připojena ještě aplikační mezipaměť. Dotazy z vrstvy `CRUD` nejprve směřují do této mezipaměti a pouze pokud nelze nalézt odpovídající výsledek v mezipaměti, je dotaz delegován na vrstvu nižší, tedy do databáze pomocí `Ujorm`.

Služby

Služby jsou pomyslným jádrem celé vrstvy. Obsahují logiku aplikace, provádí výpočty a komunikují s vrstvou *CRUD* pro získání data z databáze.

Řadiče

Serverové řadiče slouží pro odchytávání klientského asynchronního volání a dále pro překlad klientských objektů na serverové a naopak. Další úlohou řadičů je dočítání dodatečných informací pro sestavení žádostí (*requests*) službám.

[2] Java Database Connectivity
[3] ORM - Objektově relační mapování (Object Relation Mapping)
[4] Refactoring - proces, při kterém se mění struktura kódu

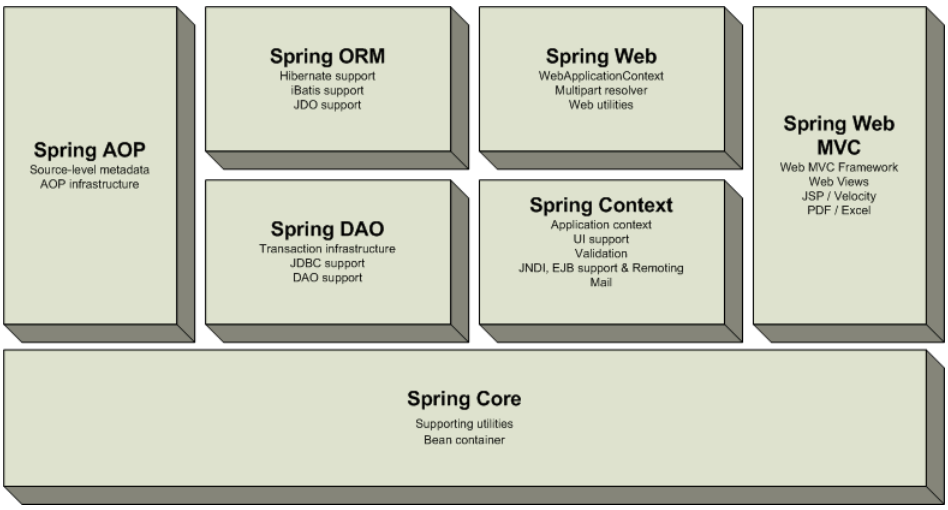
Spring

Spring je odlehčený aplikační rámec, který je využíván zejména pro tvorbu webových aplikací. Motivací rámce je usnadnění a zefektivnění vývoje podnikových aplikací, především pak při řešení běžných problémů. Spring ulehčuje práci z následujících důvodů:

1. Odstranění těsných programových vaze jednotlivých *POJO* objektů a vrstev pomocí návrhového vzoru *obrácené řízení* (Inversion of Control) a *vkládání zavilostí* (Dependency Injection).
2. Usnadnění používání a psaní unit testů.
3. Možnost volby byznys vrstvy pro aplikační architekturu.
4. Sjednocení konfigurací
5. Podpora implementace komponent pro přístup k datům.

Rámec Spring je postavený na návrhovém vzoru **obrácené řízení** (Inversion of Control). Podstatou tohoto vzoru je umožňuje uvolnit vztahy mezi těsně svázanými komponentami a přenést řízení programu z kódu na aplikační rámec. Speciálním variantou obráceného řízení je vzor **vkládání zavilostí** (Dependency Injection), který řeší vlastním způsobem konstrukci závislostí mezi jednotlivými objekty.[19] [20]

Architektura se skládá ze sedmi modulů (Core, AOP, DAO, Context, Web MVC, Web, ORM). Tyto moduly jsou znázorněny na obrázku 6.2.



Obrázek 6.2: Moduly rámce Spring (převzato z [21]

Core modul tvoří základ celého rámce. Základem tohoto moduluje tzv. *BeanFactory*, která je zodpovědná za správu objektů.

AOP modul implementuje podporu pro aspektově orientované programování. Umožňuje oddělit části kódu, které se prolínají celou aplikací (logování, transakce, autorizace).

ORM modul je určený k podpoře integrace ORM rámců.

Web modul je určený k podpoře integrace s web aplikačními rámci.

DAO modul poskytuje abstraktní vrstvu pro práci s JDBC aplikačním rozhraním.

Web MVC představuje implementaci vzoru MVC pro webové aplikace.

6.2.3 Prezentační vrstva

Prezentační vrstva se stará o zobrazení dat uživateli. Často se setkáváme s touto vrstvou v podobě grafického uživatelského rozhraní.

Pro implementaci prezentační vrstvy jsem si vybral *Google Web Toolkit* s jeho nadstavbou *Sencha GXT* a to především proto, že už jsem s těmito technologiemi setkal. Alternativou pro Google Web Toolkit se nabízí například *OpenXava*[15] nebo *JSF*[11], pro Sencha GXT se nabízí alternativy jako *Vaadin*[25], *SmartGwt*[18], *GWTruts*[5] a další.

Google Web Toolkit

Google Web Toolkit (dále jen GWT) je svobodný (open source) nástroj pro vytváření a optimalizaci komplexních webových aplikací. Cílem toho nástroje je především zvýšení produktivity vývoje a zvýšení výkonnosti těchto aplikací a to i pro vývojáře, kteří nejsou odborníky s vývojem komplexních webových aplikací a technologiemi jako je XMLHttpRequest a JavaScript. Aplikace napsané v GWT jsou kompatibilní se všemi prohlížeči, jelikož GWT automaticky generuje JavaScript kód vhodný pro každý prohlížeč. GWT je možné rozdělit do těchto tří hlavních částí:

Překladač Javy do Javascriptu je nejdůležitější částí samotného GWT. Umožňuje vývojářům psát kód webové aplikace v jazyce Java, který je následně překládán do optimalizovaného Javascriptu. Přeložený Javascript je poté použit pro interpretaci ve webovém prohlížeči.[4]

Knihovna pro emulaci JRE napodobuje činnost podmnožiny běhových Java knihoven. Je zde provedena reimplementace základního Java API[5] v JavaScriptu. Díky tomu je možné přeložený JavaScript spouštět v prostředí webového prohlížeče. Podporované knihovny jsou `java.lang`, `java.math`, `java.io`, `java.sql`, `java.util`, `java.annotation` a `java.logging`. [26] [4]

Knihovna pro vytváření uživatelského rozhraní zahrnuje mnoho dílčích částí. Jednou z nich jsou grafické komponenty. Tyto komponenty se dělí do dvou hlavních skupin - ovládací prvky (*wigets*) a panely (*panels*). Ovládací prvek je základní grafická komponenta pro tlačítka, vstupní pole atd. Panely označují skupinu komponent pro definici pozicování komponent na stránce, tedy rozložení aplikace. Dále je zde správa historie nebo podpora RPC.[26]

RPC je zkratkou pro *Remote Procedure Call*. RPC je mechanismus, který dovoluje klientskému kódu přímo spouštět metody na straně serveru.[23]

Součástí GWT je také zakomponovaná podpora asynchronního vzdáleného volání, která využívá výše zmínění RPC. Asynchronní podoba komunikace se serverem je z hlediska vývoje webových aplikací velmi podstatná. Umožňuje nám zvýšit plynulost celé aplikace. Pří použití asynchronního volání aplikace „nezamrzne“ při čekání na odpověď od serveru, ale jakmile dostane požadovaná data, provede jejich interpretaci (například překreslení grafické komponenty).

Pro usnadnění práce s GWT doporučuje společnost *Google* využít návrhový vzor MVP nebo MVC.

Vzor MVC

Návrhový vzor, který vznikl především pro potřeby rozdělení aplikací na logické části. Při rozdělení aplikace na logické části docílíme zajímavých vlastností. Především je jednodušší

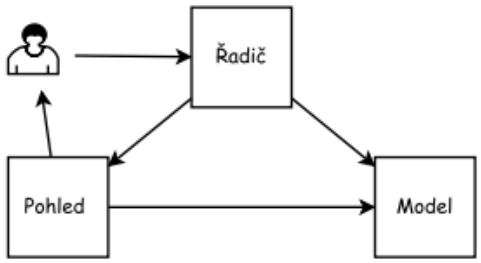
[5] Aplication Programing Interface - rozhraní pro programování aplikací

vyvíjet aplikaci v týmu, dále je možné lépe testovat jednotlivé části a také se tímto stává aplikace lépe udržovatelná a znovupoužitelná. Vzor MVC rozděluje aplikaci na tyto tři části:

Model je hlavní logická vrstva aplikace. Model se stará o přístup k datům, provádí výpočty, validace, autorizace a další. Model má pevné rozhraní, které zpřístupňuje pro další vrstvy aplikace. Jelikož model nemá informace o vrstvě, která vyžaduje jeho služby, není v jeho moci formátovat tato data. V podstatě transformuje vstupní parametry na výstupy.

Pohled („View“) se stará o zobrazení výstupů uživateli. Pohled komunikuje s modelem, který získává potřebná data k zobrazení a poté je předá pohledu, který převede data do podoby vhodné k prezentaci uživateli. Pohled také komunikuje s řadičem, kterému předává informace o událostech vyvolaných uživatelem, například vyplnění řádku tabulky, kliknutí na obrázek atp.

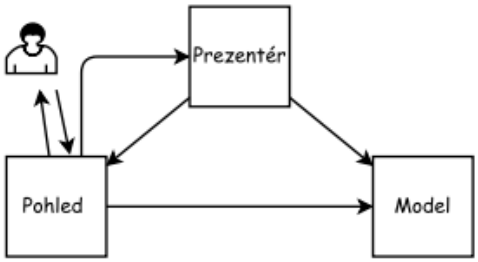
Řadič („Controller“) reaguje na události vyvolané uživatelem. Například vyvolá aktualizaci modelu po smazaní některé buňky tabulky atp.[13]



Obrázek 6.3: Struktura MVC

Vzor MVP

Návrhový vzor, který vychází ze vzoru MVC. Mají spolu podstatnou část společnou. Rozdíl ale nastává ve vrstvě prezentér/řadič. Zatímco řadič se stará o obsluhu několika *pohledů*, tak prezentér komunikuje s jedním *pohledem*. V MVC docházelo k interakci uživatele jak s *pohledem*, tak s *řadičem*. Ve vzoru MVP prezentér tuto funkci neplní.



Obrázek 6.4: Struktura MVP

Sencha GXT

Sencha GXT je komplexní Java rámec pro tvorbu webových aplikací jak pro stolní počítače, tak pro mobilní zařízení. Využívá GWT překladače. Tím zajišťuje vývojářům možnost psát aplikace v jazyce Java a překládat je do optimalizovaného HTML5 kódu.

Sencha GXT poskytuje velké množství grafických komponent, mezi které patří grid (mřížka), okna, dialogy, formuláře, menu, stromy, seznamy atd. Důležitou roli zde hraje také komunita okolo Sencha GXT, která vytváří významné množství zajímavých a specializovaných komponent. Tyto grafické komponenty uživatelského rozhraní jsou schopny spolupracovat s nativními GWT komponentami, šablonami a s grafickým správcem rozvržení, což poskytuje vývojářům kompletní podporu nad zobrazováním obsahu uživateli. Vzhled ale netvoří pouze rozvržení stránky s vhodně rozmístěnými komponentami. Pro vlastní a unikátní vzhled má zde Sencha GXT připraven tvůrce vzhledu, který používá jednoduchý konfigurační systém pro kompletní tvorbu motivů a témat.[17]

Nevýhodou tohoto rámce je jeho licence. Sencha GXT je totiž komerční projekt a pro vlastní potřebu lze použít pouze verzi s „všeobecně veřejnou licencí“ (General Public Licence-GPL). Tato verze není často aktualizována a obsahuje stále velké množství chyb. Pro výslednou aplikaci jsem využil verze `4.0.0-gpl`.

Kapitola 7

Implementace

Kapitola popisující strukturu výsledné aplikace, implementační řešení klíčových nebo netriviálních vlastností, komunikaci mezi vrstvami a grafické uživatelské rozhraní. Kapitola dále popisuje využití některých technologií pro usnadnění vývoje. Součástí kapitoly jsou fragmenty kódy demonstrující konkrétní řešení dané problematiky.

7.1 Pomocné nástroje

7.1.1 Vývojové prostředí

Používání vývojových prostředí je v dnešní době běžné. Jejich výhodou je komplexnost a způsob usnadnění vývoje. Vývojová prostředí mají často integrovanou syntaktickou nápovědu daného jazyka, integraci s databází, integraci verzovacích systémů a především podporu různých aplikačních rámců. Pro vývoj Java aplikací zmíním tři nejznámější a nejpoužívanější prostředí.

Eclipse

Vývojové prostředí s podporou značného množství programovacích jazyků (*Java*, *PHP*, *C* a další). Eclipse je ditribuován pod svobodnou licencí (*open source*) a není svázán operačním systémem (tzv. multiplatformní). Filozofií Eclipse je umožnit rozšiřování prostředí pomocí zásuvných modulů.[1]

Netbeans

Netbeans patří stejně jako Eclipse mezi multiplatformní a volně distribuované vývojové prostředí. Nabízí podporu pro programovací jazyky jako je *Java*, *PHP*, *C++* a jiné. Netbeans jsou vhodnou volbou pro malé projekty a začínající programátory. Při větších projektech, jako je například výsledná aplikace, se vývojové prostředí zpomaluje a má problémy při indexování změn. Výhodou Netbeans je jeho komunita, která vytváří velké množství zásuvných modulu a také jeho svobodná licence.[14]

Intellij IDEA

Komerční prostředí orientované na vysokou produktivitu, efektivitu a čitelnost kódu. Intellij má integrované velké množství nástrojů urychlující vývoj a především je celé prostředí svižné i při vývoji větších projektů.[9]

Z počátku jsem aplikaci vyvíjel v prostředí Netbeans, ale poté se ukázalo, že toto prostředí se po nějaké době začne zpomalovat. Z toho důvodu jsem přešel na komerční nástroj IntelliJ IDEA, který lze pro volně šířené aplikace použít zdarma, nabízí velmi dobrou podporu GWT a rámce Spring a také u něho nedochází ke zpomalování.

7.1.2 Liquibase

Při vývoji aplikace jsem narazil na problémy související se strukturou databáze na databázovém serveru, na kterém již dříve aplikace běžela. Při změně struktury databáze jsem musel tuto změnu provést jak na vývojářském databázovém serveru, tak na produkčním. Pro zjednodušení úprav databázové struktury jsem využil rámce **Liquibase**.

Liquibase je svobodný aplikační rámec sloužící pro sledování, správu a aplikaci změn schématu databáze. Velké plus zasluhuje podpora velkého množství databázových systémů. V systému je Liquibase použit pro vytváření tzv. databázových migrací. Pojmem databázové migrace značíme proces modifikace databázového schématu. Typickým příkladem může být přidání nového sloupce tabulky, vytvoření tabulky atp. Liqubase umožňuje načítat informace o změnách z textových souborů. Mezi podporované formáty patří XML, JSON, YAML a SQL. Samotná migrace je pak definována v textovém souboru pomocí klíčového slova `databaseChangeLog`.[12]

Při spuštění aplikace se spustí `Liquibase`, který zkontroluje aktuální stav databáze oproti XML definicím. V případě rozdílu se spustí aktualizace databáze. Tyto definice neobsahují jenom změny v databázi, ale i její kompletní vytvoření a naplnění výchozími daty. XML definice jsou součástí balíčku `resources.db.changelog`.

7.1.3 Unit testy

Testování je důležitou součástí vývoje softwaru. Motivací pro psaní testů je odhalení chyb, lepší pochopení kódu, usnadnění vývoje. **Unit** testování je technika sloužící pro testování menších částí kódy. Typicky pro každou část kódy by měl existovat test.

Pro testování jsem využil rámce **JUnit** a snažil jsem se otestovat všechny serverové metody, které jsou vystavené (viditelné) pomocí definovaného rozhraní. Testování klientských metod probíhalo podle příkladu užití (anglicky *use case* a to především z důvodu, že klient obsahuje převážně grafické uživatelské rozhraní, které není pomocí JUni testů přívětivé testovat.

7.1.4 Maven

Nástroj Maven slouží pro správu a automatizaci sestavování aplikací. Maven je primárně určen pro jazyk Java, ale lze jej použít i pro projekty psané v jiných jazycích. Konfigurace projektu je zapsána v souboru `pom.xml` a obsahuje především základní informace o projektu a externí knihovny (závislosti). Tyto knihovny pak Maven při překladu automaticky stáhne z repositáře a uloží do lokálního repositáře. Tím je zajištěno, že budou všechny knihovny potřebné pro překlad součástí projektu a kompilace proběhne úspěšně. Definici závislosti demonstruje následující fragment ze souboru `pom.xml`.

```
<dependency>
        <groupId>org.ujorm</groupId>
        <artifactId>ujo-core</artifactId>
        <version>1.55</version>
</dependency>
```

7.2 Struktura aplikace

Rozdělení aplikace vychází z návrhu. Každá vrstva architektury má svůj balíček, ve kterém se na nachází dílčí specifičtější podbalíčky. Mezi hlavní balíčky patří **Server** a **Client**, které reprezentují rozdělení na síťové vrstvě – architektura klient-server.

Server balíček obsahuje serverové objekty, implementaci logiky, komunikaci s databází.

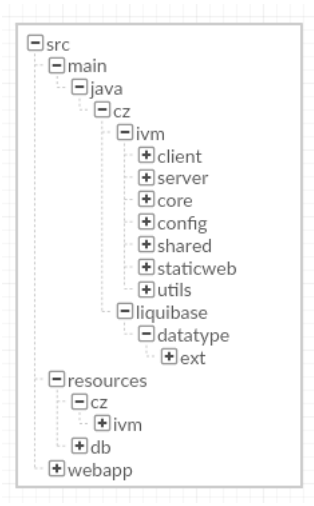
Client balíček obsahuje především implementaci grafického uživatelského rozhraní a klientské objekty.

Mezi těmito dvěma balíčky se pak nachází dva společné, **Shared** a **Utils**.

Shared („sdílený“) obsahuje především výčtové typy a konstanty.

Utils („nástroje“) obsahuje pomocné metody, například pro práci se soubory.

Posledním významným balíčkem je **Staticweb** („neměnný“). Ten je určen pro ikony, kaskádové styly, obrázky a podpůrné scripty. Základní strukturu znázorňuje obrázek 7.1.



Obrázek 7.1: Struktura výsledné aplikace

7.3 Integrace rámce Spring

Před použitím rámce Spring ve vlastním kódu je důležité vytvořit potřebné konfigurace. Primárním konfiguračním souborem je `web.xml`. V tomto souboru se bude nacházet vstupní bod (entry point) pro inicializaci Spring kontextu. Pro výslednou aplikaci jsem použil `XmlWebApplicationContext`, který umožňuje načítat konfigurace ze xml dokumentů. V těchto xml dokumentech budou vytvořena konkrétní nastavení.

Hlavní konfiguraci obsahuje dokument `spring-server.xml`. Tato konfigurace obsahuje nastavení umožňující vytvářet *beany* pomocí anotací. **Bean** je označení pro objekt, který je inicializován, sestaven a dále spravován Spring kontejnerem[19]. Tato technika vychází ze vzoru *obrácené řízení* (Inversion of Control). Tyto objekty typu *bean* lze pomocí tohoto nastavení automaticky vkládat do kódu pomocí anotace `@Autowired`. Objekty, které lze vkládat pomocí této anotace, se značí anotací `@Component`. V aplikaci jsou takto označeny služby a serverové řadiče a o správu těchto objektů a řešení závislostí mezi nimi se tedy stará Spring.

Používání vkládání závislostí je v kódu využíváno velmi často. Hlavním přínosem této metody vidím v jednoduchosti a přehlednosti zápisu.

7.4 Klientské a serverové objekty

V klient-server aplikacích je osvědčenou praxí (*best practicies*) používat pro každou z těchto vrstev individuálních objektů i přesto, že by měli být objekty na první pohled totožné. Tím je striktně odděleno, které objekty lze v dané vrstvě používat a které ne.

Aplikace využívá objektově orientovaného mapování pomocí aplikačního rámce *Ujorm*. Všechny databázové entity musí mít definici jak na serverové tak na klientské straně. Platí tedy, že jedna třída je ekvivalentem databázové entity a výčet proměnných (tzv. property) musí obsahovat všechny atributy entity. Tyto mapované objekty se nazývají *byznys objekty* a jsou umístěné v podbalíčku `*.bo`. Při definici těchto objektů lze využít anotací pro upřesnění některých vlastností. Z pohledu byznys objektů je klíčové určit pro jednotlivé proměnné, zda se jedná o konkrétní sloupec databáze (`@Column`), jestli musí být proměnná definována (`mandatory`), nebo jestli není proměnná pouze pomocná (`@Transient`).

Pro všechny **serverové objekty** existuje předek `IvmUjo`, který rozšiřuje `OrmTable-Synchronized` z balíčku `org.ujorm.implementation.orm`.

Klientské objekty mají téměř podobný zápis, avšak při definici není třeba pomocí anotací uvádět jestli daná proměnná reprezentuje atribut entity (sloupec databáze), nebo jiné vlastnosti, a to proto, že klientské objekty jsou transformované (přeložené) verze serverových objektů. Klientské objekty jsou pojmenovány téměř stejně jako jejich serverový vzor, ale přidávají k názvu předponu **C** (Client), například *CUser*, *CCourse*.

Při implementaci si však nevystačíme pouze s byznys objekty, potřebujeme i nějaké pomocné (anglicky *auxiliary*). Tyto objekty jsou umístěny v podbalíčku `*.ao` a vycházejí z třídy `IvmAuxiliaryUjo` pro serverové a `IvmAuxiliaryCujo` pro klientské. Rozdílem oproti byznys objektům je, že nepotřebují jednoznačný identifikátor, nejsou trvalá a musíme je vytvářet při každém spuštění, zatímco byznys objekty načítáme z databáze.

7.5 Překlad objektů

Jelikož aplikace obsahuje dva typy objektů, jeden pro serverové a jeden pro klientské, lze tyto objekty používat pouze na jedné vrstvě, je třeba tyto objekty mezi vrstvami transformovat, neboli překládat.

Překlad objektů zajišťuje třída `IvmControllerImpl`. Ta využívá k překladu existujícího řešení v aplikačním rámci *Ujorm* a to konkrétně v třídě `UjoTranslator`. K této implementaci je přidán překlad tzv. *map*, které slouží pro upřesnění dotazů (řazení objektů, dodatečné podmínky). Příkladem takové mapy je `LgMap`, která je využita pro definování způsobu zobrazení dat v obecném `IvmLiveGrid`.

Z tohoto důvodu, aby bylo možné přeložit jeden typ objektu na druhý musí překladač vědět, která třída na straně klienta odpovídá požadované serverové a naopak. Definice těchto dvojic je vytvořena v třídě `ServerClassConfig`.

7.6 Komunikace s databází

V návrhu aplikace jsem popsal, že dotazování do databáze je realizováno pomocí rámce `Ujorm`. Ujorm obsahuje třídu `Criterion` (kritérium), která slouží pro zjednodušení zápisu

dotazů. Jednoduchým příkladem může být výběr uživatele s identifikátorem „-1“, který lze realizovat pomocí rámce následujícím způsobem: `Criterion.where(User.id,-1)`. Kritérium pak slouží jak vstupní parametr ve službách, které komunikují s vrstvou CRUD a pro načtení záznamu pak stačí zavolat metodu `load` s tímto parametrem.

Další výhodou takovéhoto zápisu je, že lze kombinovat více kritérií a vytvářet tím komplexnější dotazy.

Nevýhodou těchto kritérií je, že se nehodí pro velmi složité dotazy nebo pro dotazy, kde se využívá seskupení více tabulek, pomocí deklarace `LEFT JOIN`. Pro takové případy existuje v aplikaci třída `UserSqlTemplate` ve které lze definovat šablonu vlastního SQL dotazu. Tato šablona může obsahovat zástupné značky, které lze za běhu aplikace nahrazovat. Příkladem může být značka `{TABLES}`, kterou lze nahradit názvem tabulky ze které se mají záznamy vybírat. Šablony mají také vlastnost, že jsou kompatibilní s MySQL a MsSQL zápisem, přičemž stačí definovat šablonu pouze pro jeden typ zápisu.

Tyto kritéria mají také svoji klientskou verzi (`CCriterion`) kterých je využito pro načítání záznamů pro zobrazení. Pro použití klientských kritérií musí zobrazovaný objekt být potomkem třídy `CrudLiveGridPanel` z rámec `Ujorm`, poté stačí pomocí metody `setCriterion` nastavit požadovaná kritéria a při načítání dat je toto kritérium použito. Klientská kritéria jsou také využita v kombinaci s filtry. Při změně hodnoty filtru (často definovaného jako `ComboBox`) se přenastaví kritérium a potřebné tabulky jsou překresleny. Nevýhodou používání kritérií na klientovi je nutnost přidávat dodatečné informace pro serializaci a tím se zápis stává složitějším a hůře čitelným.

7.7 GUI

Grafické uživatelské rozhraní využívá GWT a jeho nadstavbu Sencha GXT. O spuštění a inicializaci se stará třída `Application`, která implementuje rozhraní `EntryPoint` (vstupní bod) s jedinou metodou `onModuleLoad`. Při volání této metody se vytvoří kořenový kontejner `ViewPort`, který slouží pro zobrazování pohledů. Výsledná aplikace obsahuje pouze jeden pohled, který je definován třídou `ApplicationPanel`. Tento panel je tvořen dvěma částmi. První část tvoří `SectionTabPanel`, který umí vytvářet záložky (sekce). Tyto sekce vytváří menu aplikace. Druhou částí je spodní (bottom) panel sloužící pro odhlášení uživatele (viz obrázek 7.2).

Pro vytvoření nové sekce je třeba vytvořit třídu rozšiřující `Section` a následně zaregistrovat tuto sekci v `ApplicationPanel` pomocí metody `addSection`. Nově vytvořená sekce musí implementovat metodu `createTabContent`. Tato metoda je tzv. *líná* (lazy) a je volána až při samotném zobrazení grafické komponenty. V důsledku to znamená, že inicializace potřebných objektů se provede, až když je komponenta vykreslována, namísto inicializování v konstruktoru. Tím se zvyšuje rychlost aplikace.

7.7.1 Zobrazení dat

Základní třídou pro zobrazení dat v aplikace je `IvmLiveGrid`. Tato třída rozšiřuje `CrudLiveGridPanel` z balíčku `org.ujorm.gxt.client.gui.livegrid`, který má vytvořenou obecnou implementaci pro základní operace (CRUD) a zobrazování klientských objektů v tabulce. Pro zobrazení dat je požadované vytvořit konfiguraci pohledu. Tato konfigurace se skládá z následujících kroku:

- V třídě `LiveGridKeys` vytvořit identifikátor podhledu.

Obrázek 7.2: Rozvržení komponenty ApplicationPanel

- Vytvoření databázové migrace v souboru `table_header_config.xml` pro pohled.
- Vytvoření databázové migrace v souboru `table_column_config.xml` pro zobrazované položky pohledu (jednotlivé proměnné zobrazované třídy).
- Nový `IvmLiveGrid` s parametry (identifikátor pohledu, třída zobrazovaného objektu, název databázové entity/tabulky).

Protože zde vytváříme databázové migrace pro pohled a jeho položky zobrazení, musíme pro ně rozšířit strukturu databáze. Nové tabulky se jmenují `table_header_config` a `table_column_config`.

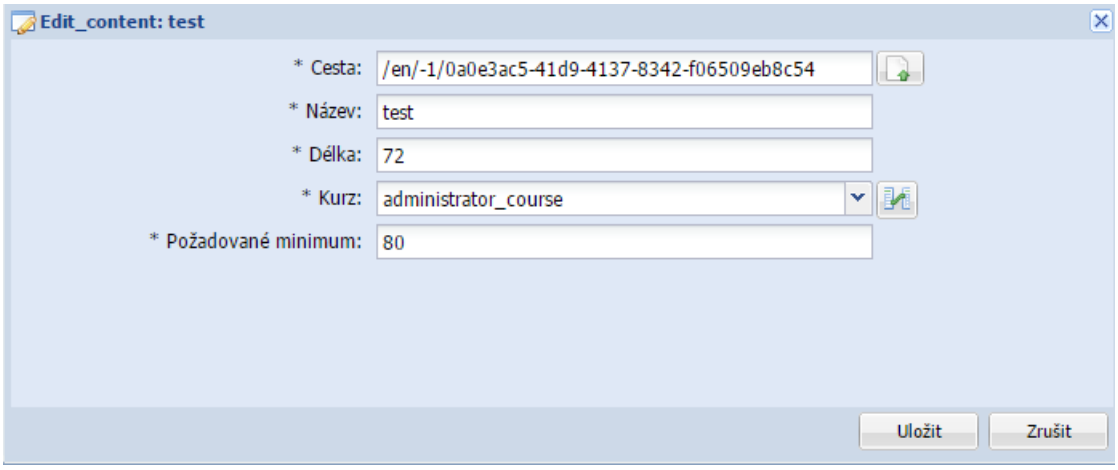
7.7.2 Vytváření a úprava byznys objektů

Pro vytváření a úpravu byznys objektů jsem implementoval editační dialog (`IvmEditDialog`), který je navržen pro klientské byznys objekty (`IvmCujo`). Editační dialog vychází z rámce `Ujorm` a jeho třídy `EditDialog`. Klíčovým parametrem konstruktoru dialogu je upravovaný nebo vytvářený objekt. Při vytváření nového objektu se nejdříve vytvoří prázdny objekt s parametrem `new` (`new = true`) a následně se předá v parametru konstruktoru editačního dialogu. Poté je úprava a vytváření velmi podobná. Editační dialog si získá jednotlivé proměnné (property) objektu a podle jejich typů vytvoří nový řádek. Z těchto řádku je poté tvořen celý dialog. Samotné uložení, vytvořeného nebo upraveného objektu do databáze je realizováno asynchronním voláním pomocí obecného řadiče (`CrudController`).

7.8 Interaktivní videa

7.8.1 Zobrazení videa

Pro zobrazení interaktivního videa jsem vytvořil vlastní komponentu `ContentComponent`. Vstupními parametry při vytváření komponenty je klientský objekt typu `CContent`. Zobrazování samotného videa obstarává upravená H5P knihovna, *h5p-standalone*[]. Tato knihovna je upravená pro zobrazení interaktivního videa pouze ve webovém prohlížeči bez serverové podpory. Obsahuje všechny potřebné H5P knihovny, které jsou sjednocené v JavaScript



Obrázek 7.3: Úprava záznamu videa pomocí editačního dialogu

souboru `h5p-standalone-frame.js`. Pro samotné zobrazení videa je třeba vytvořit HTML prvek s atributem třídy `h5p-container` a spustit následující skript.

```
(function($) {
    $(function() {
        $('.h5p-container').h5p({
            frameJs: 'cesta k h5p-standalone-frame.js',
            frameCss: 'cesta k h5p.css',
            h5pContent: 'cesta k videu'
        });
    });
})(H5P.jQuery);
```

Vytvoření HTML prvku a předešlý skript jsem integroval do zmíněné GXT komponenty `ContentComponent`. Ovšem to ke korektnímu zobrazení nestačí, jelikož je třeba ještě před vykreslením komponenty načíst kaskádové style `h5p.css`. Načtení kaskádových stylů jsem přidal do hlavní HTML stránky `Application.html`. Pro zobrazení videa stačí skriptu předat požadovanou cestu nebo odkaz. Cesta k samotnému videu by měla směřovat na datové skladiště, které je součástí serveru. Pro získání videa s požadovanou cestou (adresou) jsem využil tzv. *servlet*[1].

V aplikaci to pak funguje následovně: Cesta k videu je pouze relativní a obsahuje předem definovaný zápis. Při vytvoření HTTP dotazu s použitím zápisu určeného pro servlet je tento požadavek zachycen a zpracován. O zpracování a zachycení požadavku je zodpovědný registrovaný servlet.

Vytvoření servlet třídy s konfigurací vysvětlím na konkrétním příkladu pro stahování videí ze serveru.

- Vytvořit třídu rozšiřující `FileDownloadServlet`.
- Implementovat metodu `getFilePath`. Metoda vrací cestu k videu uloženého na serveru.
- Registrovat servlet v souboru *web.xml*.

[1]Servlet - je nástroj pro zpracování HTTP požadavků

Servlet registrace

Nejdříve je třeba vytvořit definici `servlet` s parametry `servlet-name` a `servlet-class`. `Servlet-name` definuje název pro servlet, `servlet-class` odkazuje na třídu, které bude dotaz delegován. Dále vytvoříme mapování URL pomocí `servlet-mapping` s parametry `servlet-name` a `url-pattern`. V tomto případě `servlet-name` odkazuje na název definovaného servletu a `url-pattern` definuje URL daného HTTP požadavku, který má být pomocí servletu mapován.

Konkrétně pak pro stahování videa vypadá definice následovně:

```
<servlet>
    <servlet-name>getContent</servlet-name>
    <servlet-class>
        cz.ivm.server.servlet.ContentFileDownloadServlet
    </servlet-class>
</servlet>
    <servlet-mapping>
    <servlet-name>getContent</servlet-name>
    <url-pattern>/content/*</url-pattern>
</servlet-mapping>
```

Použití servletu

Samotné použití servletu využívá definice `url-pattern`, konkrétně jeho hodnota `/content`. Tato hodnota značí, že všechny HTTP požadavky, kde URL obsahuje `/content/` a nějaké další parametry , budou zpracovány pomocí `ContentFileDownloadServlet`. Proto stačí do skriptu, který se stará o vykreslení videa, přidat do proměnné `h5pContent` mapované URL (`/content/`). K této URL je následně třeba přidat jednoznačný identifikátor videa, aby bylo možné načíst požadovaná data na straně serveru.

7.8.2 Uživatelské interakce

Pro zachycení uživatelských interakcí aplikace využívá H5P aplikačního rozhraní (xApi)[7]. Samotné navěšení na *xApi* je realizováno pomocí externího odesílatele (external dispatcher).

Ukázka navěšení na xAPI: [7]

```
H5P.externalDispatcher.on('xAPI', function (event) {
            //akce
        });
```

Parametr `event` obsahuje informace o zdroji události. Tento parametr obsahuje především identifikátor objektu interaktivního videa, typ události, změněné hodnoty atd. Pro ukládání těchto akcí je třeba znát k jakému videu konkrétní interakce patří, a který uživatel provedl interakci. Je tedy nezbytné rozšířit událost.

Rozšíření zachycené události je implementováno v jazyce JavaScript a pro odesílání informací na server je využito technologie AJAX (*Asynchronous JavaScript and XML*). Implementaci výsledného skriptu lze nalézt v souboru `resources/scripts/evaluationScript.js`

Zachycení na straně serveru jsem realizoval pomocí REST rozhraní, které je integrováno v *Spring* rámci. Rozhraní je zapotřebí registrovat podobně jako `Servlet` v souboru `web.xml`. V tomto případě ale bude parametr `servlet-class` obsahovat hodnotu `org.springframework.web.servlet.DispatcherServlet`.

V Java kódu pak pomocí Spring anotací je nezbytné vytvořit REST řadič, který bude obsahovat metodu mapovanou na požadované URL. Definování řadiče lze zajistit Spring anotací `@RestController` a pro mapování metody je definována anotace `@RequestMapping`.

Následující kód demonstruje vytvoření AJAX dotazu.

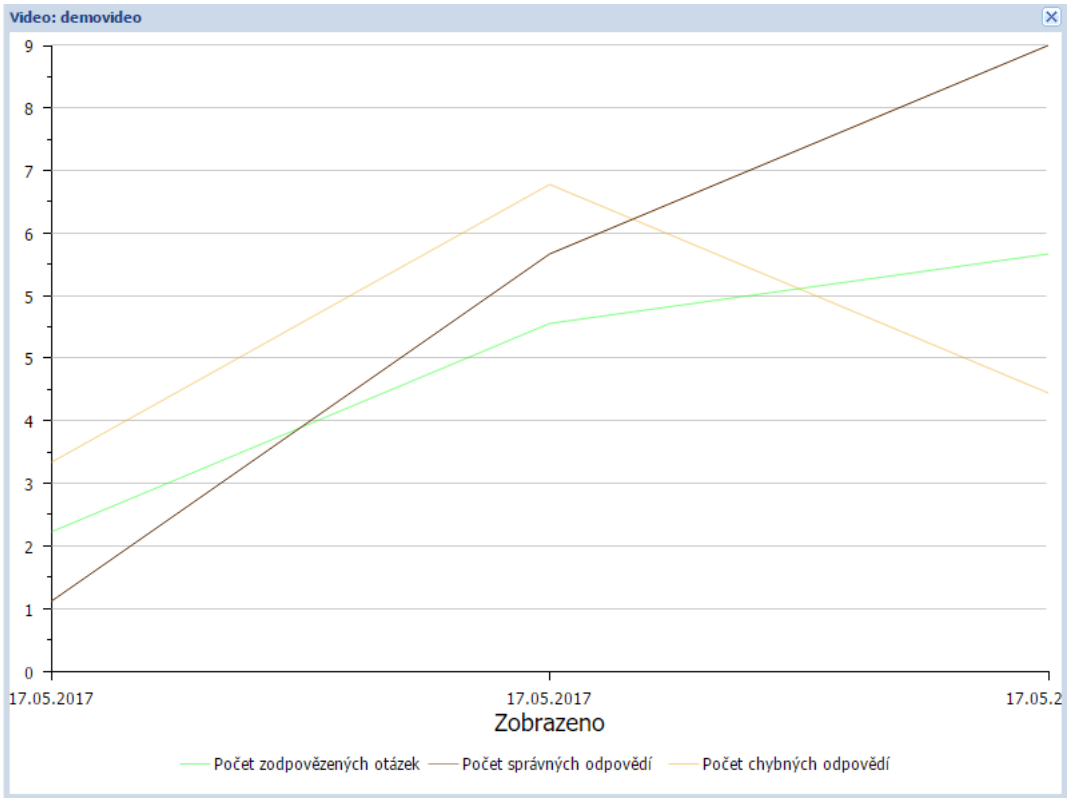
```
$.ajax({
    url: 'rest URL',
    type: 'POST',
    data: odesilana data,
    dataType: 'json',
    contentType: "application/json; charset=utf-8",
    success: function(result) {}
});
```

7.8.3 Statistické údaje a grafy

Aplikace umožňuje zaznamenávat jednotlivé uživatelské interakce, ze kterých získává potřebná data, která by měla sloužit pro vytváření statistik. Bohužel data, která jsou odesílána při interakci jsou velmi strohá a tak aplikace uchovává pouze počet správně a chybně „vytvořených odpovědí“ a počet otázek, na které bylo v rámci zhlédnutí odpovězeno.

Spojením „vytvořené odpovědi“ je myšlena konkrétní uživatelská odpověď na otázku, jelikož otázka může mít více odpovědí a uživatel odpovídá právě na konkrétní odpověď. Při implementaci bylo důležité vyřešit některé neočekávané stavy, které video přehrávač H5P podporuje. Při přehrávání videa je totiž možné některé z otázek opakovat a pokud by se zaznamenávaly všechny odpovědi jako nové, došlo by k nesrovnalostem v počtu otázek videa a počtu uživatelských odpovědí. Z tohoto důvodu se jednotlivé odpovědí pouze aktualizují.

K těmto datům jsem přidal grafy, které lépe znázorňují vývoj uživatelských odpovědí v závislosti na čase. Zobrazení grafů je vytvořeno pomocí modulu `gxt-charts`, který je součástí rámce Sencha GXT.



Obrázek 7.4: Graf zobrazující historii zhlédnutí videa uživatele

7.8.4 Vytvoření interaktivního videa

Interaktivní video lze vytvořit pomocí rámce H5P. Rámec totiž obsahuje video editor pro vytváření interaktivních videí. Po vytvoření videa lze toto video i s potřebnými knihovnami a daty stáhnout ve formátu `.h5p`. Tento formát je pouze přejmenovaný archivační formát `.zip` a proto není třeba k otevření balíku speciálního softwaru. Pro nahrání videa na aplikační server budeme potřebovat konkrétní `.h5p` archiv s videem a potřebnými daty.

Pro nahrání videa jsem použil stejného principu jako pro stahovaní (7.8.1). Vytvořil jsem obecný abstraktní servlet pro nahrávání souborů (`FileUploadServlet`), který jsem dále modifikoval pro formát `.h5p`. Pro výslednou třídu sloužící pro nahrání archivu (`ContentFileUploadServlet`) je ještě zapotřebí vytvořit mapování v souboru `web.xml`. Součástí této třídy je také implementace rozbalení archivu. Výsledná struktura po rozbalení odpovídá požadované struktuře pro zobrazení v komponentě `ContentComponent`.

Z pohledu uživatelského rozhraní se při vytváření interaktivního videa v aplikace zobrazí prázdný editační dialog obsahující jednotlivé proměnné (property) objektu. Zásadní pole tohoto dialogu je `url`, které obsahuje tlačítko pro nahrání videa. Po kliknutí na tlačítko nahrát se otevře `UploadDialog`, který obsahuje obecnou implementaci nahrávání souborů na server. Klíčovým parametrem je zde `postUrl`, který reprezentuje url mapovaný servletem. Po nahrání videa je uživateli zobrazena informace o výsledku operace. V případě úspěchu lze dialog uložit a proběhne samotné uložení záznamu do databáze.

Ještě před uložením objektu jsou na serveru provedeny validace, které kontrolují především duplicitu h5p souborů, jelikož by se mohlo stát, že ukládáme jedno video vícekrát a při

dotazovaní by tak nebylo jednoznačné, se kterým videem chceme pracovat. Tuto kontrola provádí služba `H5PService`.

Proběhnou-li validace bez chyby, je možné vytvořit záznam v databázi. V tomto momentě server provádí ještě jednu činnost. Z nahrávaného videa získá soubor `content.json`, který obsahuje veškerá data sloužící pro přehrávání videa (otázky, odpovědi, čas zobrazení a další). Tato potřebná data získá a vytvoří pro ně záznamy v databázi.

Pro podporu vytváření interaktivních videí jsem vytvořil vlastní správu a nahrávání video souborů typu `.mp4` nebo `.webm`. Video soubor je možné nahrát na server a získat k němu odkaz, který lze následně použít v H5P video editoru provozovaném na `H5P.org` serveru. Tímto způsobem lze odstranit omezení maximální povolené velikost souboru, kterou má video editor nastavenou.

Nahrávání těchto videí je implementované podobným způsobem jako nahrávání interaktivních videí. Je zde použit servlet s názvem `uploadVideo` a samotné nahrání a uložení videa obstarává třída `VideFileUploadServlet`.

Tuto podporu by bylo dobré do budoucna odstranit a nahradit plnohodnotným video editorem.

7.9 Jazyky a překlady

Pro potřeby vícejazykové aplikace je zapotřebí mít vlastní překladač. V aplikaci jsem tento překladač implementoval v třídě `Translator` implementující rozhraní `TranslatorProvider` z balíčku `org.ujorm.gxt.client.providers`. Zjednodušeně dostane překladač na vstupu řetězec a navrátí přeloženou hodnotu.

Překlady jsou uložené v databázi v tabulce `Message` a její hlavní atributy jsou `key` (klíč), `locale` (lokalizace) a `value` (hodnota). Jedna zpráva pro překlad je tedy definována pomocí klíče (řetězcový identifikátor), lokalizace (určuje v jakém jazyce je uložená hodnota zprávy) a samotno hodnotu.

Načítání těchto překladů z databáze se provádí při přihlášení uživatele. Podle poslední nastavené lokalizace se načtou potřebné překlady a uloží se do mapy (`HashMap`), ze které bude překladač získávat data. V případě potřeby přeložení klíče je zavolána metoda `translate` a pokud překladač nalezne odpovídající záznam, vrátí přeloženou hodnotu dotazovanému objektu.

Příkladem použití překladů je grafické uživatelské rozhraní. Jednotlivé grafické prvky obsahují pouze klíč pro překlad a při vykreslování je samotný překlad získán pomocí překladače a následně vykreslen.

Překlady jsou uloženy v xml definicích (`message.cs.xml`, `message.sk.xml` a `message.en.xml`) a jsou do databáze importovány při startu aplikace za použití rámce `Liquibase`.

V aplikaci existují dva druhy překladů, jeden pro uživatelské rozhraní a druhé pouze pro knihovny aplikačního rámce H5P. Překlady pro H5P se nenačítají při přihlášení, ale pouze v momentě, kdy je ukládáno nové video. V tomto okamžiku se nahraje do paměti soubor `content.json` a přeloží se požadované texty. Překladů H5P materiálů je možné si všimnout při přehrávání videa, kdy jednotlivé ovládací prvky jsou lokalizované do aktuálního jazyka, který má uživatel nastaven.

Kapitola 8

Testování

Testování výsledné aplikace proběhlo pomocí anonymního dotazníku. Uživatelům byl dočasným přístupem k aplikaci přes webové rozhraní. Celkem se zapojilo do testování 18 respondentů ve třech věkových kategoriích. Celkem 10 respondentů z kategorie 18-25 let, 4 respondenti z kategorie 26-40 let a 4 respondenti z kategorie 41-60 let. Testování proběhlo na starší verzi aplikace a velmi pomalém serveru a tyto parametry se projevily na výsledcích. Dotazovaní uživatelé měli občasné problémy s načítáním interaktivních videí, některé grafické prvky se nezobrazovaly podle očekávání a v některých případech nešlo video spustit. Dotazovaní uživatelé měli uživatelská oprávnění pouze pro zobrazování interaktivních videí a kurzů, nemohli tedy hodnotit administrátorské operace, které lze v rámci aplikace provádět. Výsledky dotazníku lze nalézt v příloze.

Další testování jsem prováděl sám na nové (výsledné) verzi aplikace a výkonném lokálním serveru. V této verzi jsem nezaznamenal žádné potíže s vykreslováním grafických prvků, zobrazováním videa, rychlostí aplikace nebo při interakci s videem a to přesto, že jsem testování prováděl ve vývojářském módu, který značně zpomaluje aplikaci.

Při delším testování jsem narazil na některé chyby, které aplikace obsahuje. Příkladem jsou některé validace na formuláři nebo špatné sestavení kritérií, které vede k chybnému zobrazení dat. Do budoucna by bylo vhodné tyto chyby z aplikace odstranit.

Kapitola 9

Závěr

Práce se zabývala tvorbou webové aplikace s interaktivními vzdělávacími videi, správou uživatelů a kurzů. Výsledná aplikace je napsána v jazyce Java.

V teoretické části této práce jsou popsány možnosti interaktivních videí, použité programovací jazyky, návrh aplikace a použité technologie.

Implementační část popisuje především strukturu aplikace a její hlavní části, jako je grafické uživatelské rozhraní a interaktivní videa. Tato část také názorně popisuje řešení důležitých vlastností aplikace jako je přehrávání a vytváření interaktivních videí nebo správa kurzů a uživatelů.

Poslední část popisuje reálné testování, které odhalilo některé chyby a pomohlo k dalšímu vylepšení aplikace.

Hlavní přínos práce spočívá v možnosti zaznamenávání uživatelských aktivit, které lze využít pro zlepšení samotného výukového materiálu nebo k hodnocení uživatelů. Aplikace je lokalizovaná ve třech jazycích (angličtina, čeština, slovenština) a je možné ji jednoduše rozšiřovat. Další vlastností výsledné aplikace je její licence, která umožňuje zdarma použít a upravovat zdrojové kódy.

Do budoucna by se mohla aplikace rozšířit o plnohodnotný editor videa, podporu většího množství H5P knihoven (například prezentace) a o další statistické funkce nebo o plánovač výuky. Dále by bylo vhodné odstranit některé zmíněné chyby obsažené v aplikaci.

Práce byl prezentována na konferenci INFOS 2017 jako nová možnost školení knihovnic a knihovníků.

Literatura

[1] *Eclipse*. [Online; navštíveno 14.05.2017].
URL https://eclipse.org/

[2] *Framework*. [Online; navštíveno 20.04.2017].
URL https://techterms.com/definition/framework

[3] *Go Java*. [Online; navštíveno 2.05.2017].
URL https://go.java/index.html

[4] *GWT project*. [Online; navštíveno 7.05.2017].
URL http://www.gwtproject.org/

[5] *GWTruts*. [Online; navštíveno 15.05.2017].
URL https://www.openhub.net/p/GWTruts

[6] *H5P*. [Online; navštíveno 8.05.2017].
URL https://h5p.org/

[7] *H5P and xAPI*. [Online; navštíveno 11.05.2017].
URL https://h5p.org/documentation/x-api

[8] *H5P documentation*. [Online; navštíveno 8.05.2017].
URL https://h5p.org/documentation

[9] *IntelliJ IDEA*. [Online; navštíveno 14.05.2017].
URL https://www.jetbrains.com/idea/

[10] *Java program execution*. [Online; navštíveno 5.05.2017].
URL https://commons.wikimedia.org/wiki/File:Java-program-execution.png

[11] *JavaServer Faces Technology*. [Online; navštíveno 15.05.2017].
URL http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html

[12] *Liquibase*. [Online; navštíveno 8.05.2017].
URL http://www.liquibase.org/documentation/index.html

[13] *MVC Overview* . [Online; navštíveno 6.05.2017].
URL https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx

[14] *Netbeans*. [Online; navštíveno 14.05.2017].
URL https://netbeans.org/

[15] *OpenXava*. [Online; navštíveno 15.05.2017].
URL http://www.openxava.org/

[16] *Playposit*. [Online; navštíveno 8.05.2017].
URL https://www.playposit.com/

[17] *Sencha GXT*. [Online; navštíveno 8.05.2017].
URL https://www.sencha.com/products/gxt/#overview

[18] *Smart GWT*. [Online; navštíveno 16.05.2017].
URL http://www.smartclient.com/product/smartgwt.jsp

[19] *Spring*. [Online; navštíveno 17.05.2017].
URL https://spring.io/

[20] *Spring Framework – představení J2EE lightweight kontejneru*. [Online; navštíveno 17.05.2017].
URL https://www.interval.cz/clanky/spring-framework-predstaveni-j2ee-lightweight-kontejneru/

[21] *Spring modules*. [Online; navštíveno 11.05.2017].
URL https://www.interval.cz/podklady/1999-2008/pichlik/1250/spring-overview.png

[22] *The Open Source Definition*. [Online; navštíveno 3.05.2017].
URL https://opensource.org/docs/definition.html

[23] *Using GWT RPC*. [Online; navštíveno 7.05.2017].
URL http://www.gwtproject.org/doc/latest/tutorial/RPC.html

[24] *Uživatelské rozhraní (user interface)*. [Online; navštíveno 5.05.2017].
URL https://wikisofia.cz/wiki/U%C5%BEivatelsk%C3%A9_rozhran%C3%AD_(user_interface)

[25] *Vaadin*. [Online; navštíveno 15.05.2017].
URL https://vaadin.com/home

[26] *Google Web Toolkit*. 9 2006, [Online; navštíveno 7.05.2017].
URL https://www.interval.cz/clanky/google-web-toolkit/

[27] *Entity-relationship diagram*. 9 2015, [Online; navštíveno 1.05.2017].
URL http://www.sallyx.org/sally/psql/erd.php

[28] Kolektiv: *PHP 5, MySQL, Apache - vytváříme webové aplikace*. Computer Press, 2006, iSBN: 80-251-1073-7.

[29] Kuda, M.: *Design of company contracts database*. bakalářská práce, Brno: University of Technology, 2015.

[30] Larsen, J.: *Get programming with Javscript*. Manning Publications Co, 2016, ISBN 971617293108.

[31] Managementmania: *Saas*. [Online; navštíveno 22.04.2017].
URL https://managementmania.com/cs/software-as-a-service

[32] Ponec, P.: *Ujorm User Guide*. 2013, [Online; navštíveno 1.05.2017].
URL http://ujorm.org/orm/tutorial/

[33] Schild, H.: *Java 7 - Výukový kurz*. Brno: Albatros, první vydání, 2012, ISBN 978-80-251-3748-2.

[34] Zendulka, J.; Rudolfová, I.: *Databázové systémy*. 7 2006, studijní opora.

Přílohy

Příloha A

Obsah CD

– ivm - složka se zdrojovými soubory, konfigurace pro Intellij idea, `pom.xml` a výchozí data.
– ivm.zip - archiv s konfigurací serveru (přístup do databáze, schéma db a další)
– repository.zip - archiv obsahující potřebné knihovny pro sestavení projektu
– demoVideo.h5p - demostrační video materiál

Příloha B

Konfigurace vývojového prostředí

Konfigurace určena pro vývojové prostředí: Intellij Idea 2017.1.2
Požadováno Java JDK a JRE 1.8+

1. Zapnout databázový server (doporučeno MYSQL).
2. Vytvořit databázové schéma "ivm"s podporou češtiny (CREATE SCHEMA ivm DEFAULT CHARACTER SET utf8 COLLATE utf8_czech_ci ;)
3. Itellij IDEA -> import -> vybrat pom.xml
4. Otevřít pom.xml -> pravé tlačítko -> Maven -> download sources
5. Exportovat obsah z repository.zip do .m2
6. Vyvořit GWT konfiguraci: Run -> Edit configurations -> add (tlačítko +) -> GWT Configuration
7. Nastavení GWT konfigurace: Module = Ivm, VM options -Xmx2048m
8. Extrahovat archiv ivm.zip do:

   Windows = `C:\\` (požadovaná struktura `C:\\ivm`)

   Linux = `/` (požadovaná struktura `/ivm`)
9. Otevřít user.xml (`C:\\ivm\user.xml` nebo `/ivm/user.xml`) a nakonfigurovat `db_password` a `db_user` pro přístup do databáze, popřípadě `db_schema`
10. V případě využití externího úložiště pro ukládání video formátů (.mp4) je třeba nastavit data.path. Tato konfigurace je využita pouze pro upload samotného videa v záložce video editor a na fungování celé aplikace nemá výrazný vliv.
11. Pro případ, kdy je požadováno spuštění JUnit testů je třeba vytvořit databázové schéma `ivm_test` (v GWT konfiguraci by nemělo nastat).
12. Spustit GWT konfiguraci

Příloha C

Manuál

C.1 Uživatelské přístupy

1. Uživatelské jméno: admin, heslo: ivmpassword
2. Uživatelské jméno: teacher, heslo: ivmpassword
3. Uživatelské jméno: user, heslo: ivmpassword

C.2 Správa kurzů a videí

1. Přihlásit se pod účtem *teacher* nebo *teacher*.
2. Záložka *správa kurzů* a vytvořit kurz.
3. Záložka *správa videí*, vytvořit. Nejdříve je nutné vyplnit název, dílku, kurz, požadované minimum a poté zvolit *nahrát soubor*. Kurz vybereme vytvořený z předešlého kroku.
4. Ve správě kurzu poté přiřadíme uživatele *user* do kurzu.

C.3 Zobrazení videa

1. Přihlásit se jako *user* nebo *admin*.
2. Záložka kurz a v přiřazeném kurzu označit a spustit video.

C.4 Zobrazení grafů

1. Přihlásit se jako *user* nebo *admin*.
2. Záložka *správa kurzů*.
3. Dvakrát kliknout na uživatele u kterého chceme zobrazit grafy.
4. Vybereme video u kterého chceme zobrazit grafy.

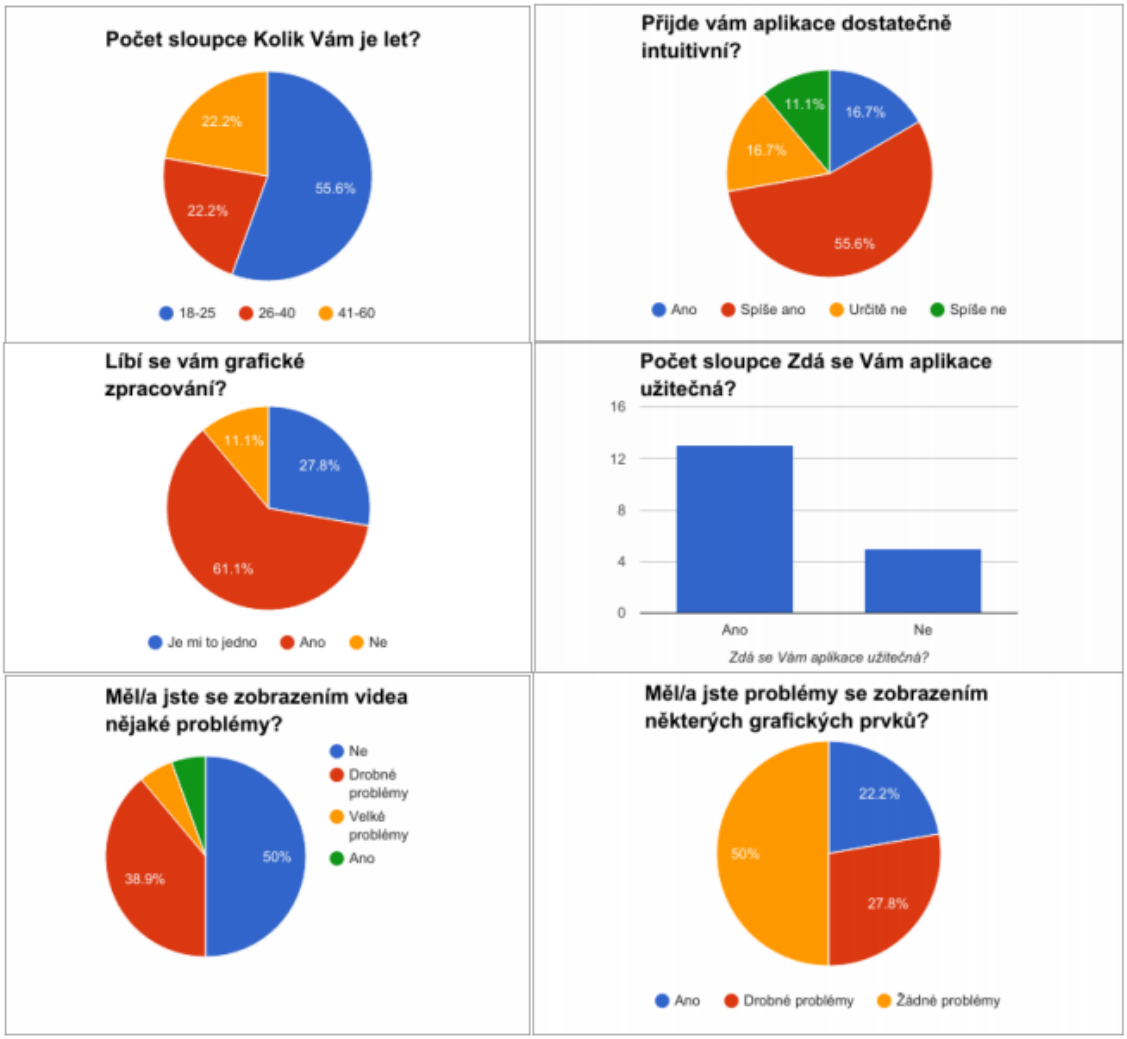
C.5 Zobrazení historie

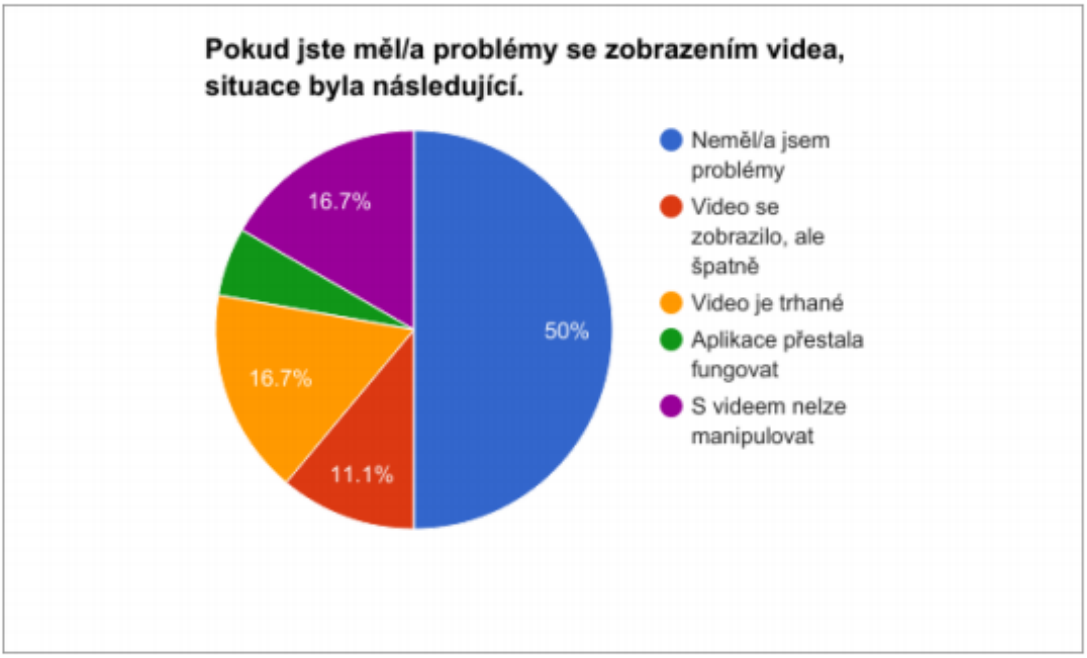
Prerekvizity: Před tímto krokem bylo alespoň jednou spuštěno video pod přihlášeným uživatelským účtem.

1. Přihlásit se jako *user* nebo *admin*.
2. Záložka *historie*.
3. Ve filtrech zadat do pole „Zodpovězených otázek v %“ hodnotu 0. Tím se zobrazí všechny výsledky.
4. Pomocí filtrů můžeme upřesnit vyhledávané výsledky.

Příloha D

Výsledky testování




Pokud jste měl/a problémy se zobrazením videa, situace byla následující.
16.7%
50%
16.7%
11.1%
Neměl/a jsem problémy
Video se zobrazilo, ale špatně
Video je trhané
Aplikace přestala fungovat
S videem nelze manipulovat